4.6 Sequential Circuit Design

We now have the necessary background and intuition to start working on the sequential circuit design process. To illustrate the concepts we focus on sequential circuits using JK flip-flops. Recall that a sequential circuit consists of a combinational circuit that produces output and a feedback circuit for state variable feedback (Figure 4.1). We use JK flip-flops for the feedback circuit design. The design process, however, can be generalized to other devices such as SR latches.

In the next two subsections, we consider designing counters using JK flip-flops. The first counter example is the 3-bit binary counter we have discussed in the last section, except that we derive a synchronous design. We use this example to introduce the design process. The following section describes how we can use this process to design a general counter.

Counters are a special case of more general sequential circuits. Section 4.6.2 describes the design process for general sequential circuits.

4.6.1 Binary Counter Design with JK Flip-Flops

We start our discussion with a simple 3-bit synchronous counter design example. In analyzing a sequential circuit, we go from input to output. For example, we might know JK inputs and we want to know the next state outputs. We use the JK flip-flop truth table for this purpose.

In designing a sequential circuit, we know the output state and we have to find the input combination. For this purpose, we build an excitation table, which can be derived from the truth table. Table 4.3*a* shows the truth table for the JK flip-flop. In this table, we have explicitly included the present output $\mathrm{Q_n}$. The excitation table consists of $\mathrm{Q_n}$ and $\mathrm{Q_{n+1}}$ as well as J and K inputs as shown in Table 4.3*b*. In this table, "d" indicates a don't care input. For example, to change the output from 0 to 1, the J input has to be 1, but the K input doesn't matter (second row in Table 4.3*b*). We use this excitation table extensively in designing sequential circuits in the remainder of this chapter.

We can now write the combination of JK inputs that take the circuit from the current state to the next state. To facilitate this process, we create a new table with several groups of columns (Table 4.4). The first group is the current state output, the second group is the next state output, and the last group consists of the JK inputs for each flip-flop in the circuit. For our 3-bit counter example, we have three bits A B C to represent the current state output and another three bits to represent the values of the next state as shown in Table 4.4. Since each bit is stored in a JK flip-flop, we have three pairs of JK inputs corresponding to the three flip-flops A, B, and C.

Filling the entries of this table is straightforward. In our example, the next state is given by (current state + 1) modulo 8. Then we will fill the JK inputs. To do this, look at the corresponding bits in the current state and next state and write down the JK input combination from the excitation table that gives the required transition from $\mathrm{Q_n}$ to $\mathrm{Q_{n+1}}$. For example, for the first row, current state and next state for the A bit is 0. Thus, we write $\mathrm{J_A} = 0$ and $\mathrm{K_A} = \mathrm{d}$ for flip-flop A. As another example, in the last row, the A bit changes from 1 to 0. Thus, JK inputs will have to be d1 (third row of the excitation table). This is the entry you see in the last row for the JK inputs of the A flip-flop.

Table 4.3 Excitation table derivation for the JK flip-flops

(a) JK flip-flop truth table

J	K	Q_n	Q_{n+1}
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	0

(b) Excitation table for JK flip-flops

Q_n	Q_{n+1}	J	K
0	0	0	d
0	1	1	d
1	0	d	1
1	1	d	0

Table 4.4 Design table for the binary counter example

Present state			Next state			JK flip-flop inputs					
A	B	C	A	B	C	J_A	K_A	J_B	K_B	J_C	K_C
0	0	0	0	0	1	0	d	0	d	1	d
0	0	1	0	1	0	0	d	1	d	d	1
0	1	0	0	1	1	0	d	d	0	1	d
0	1	1	1	0	0	1	d	d	1	d	1
1	0	0	1	0	1	d	0	0	d	1	d
1	0	1	1	1	0	d	0	1	d	d	1
1	1	0	1	1	1	d	0	d	0	1	d
1	1	1	0	0	0	d	1	d	1	d	1

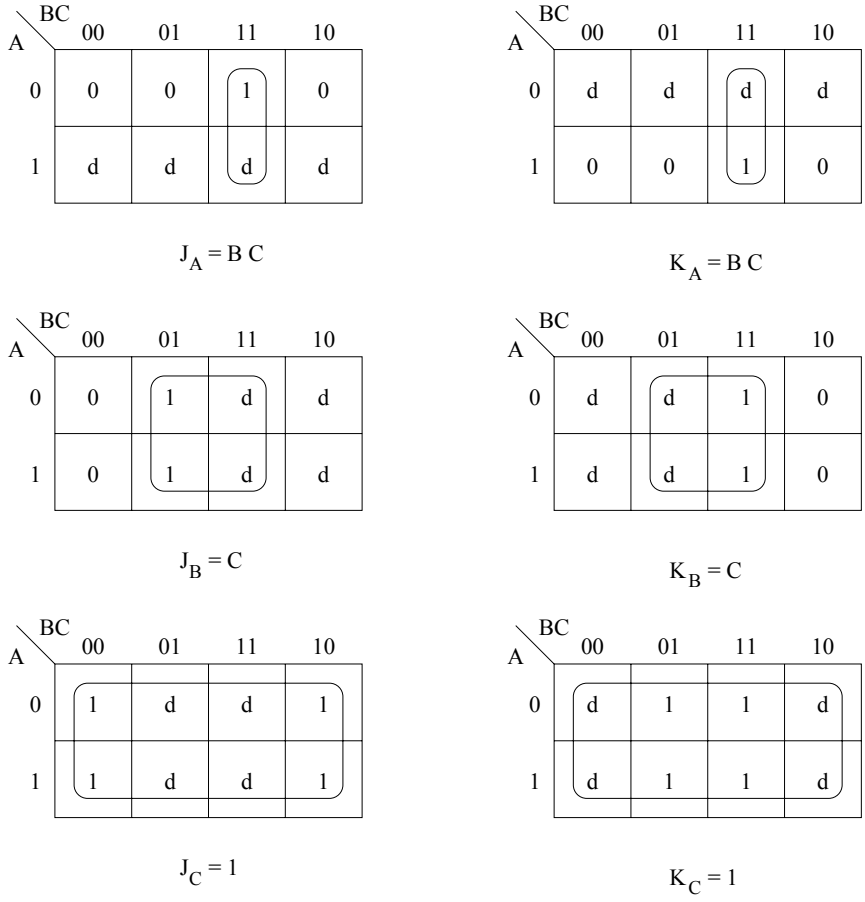

Figure 4.18 Karnaugh maps to derive simplified logic expressions for the JK inputs.

Once we have identified the JK inputs for each flip-flop, all we have to do is to see if we can simplify the logical expression for each input. This step is similar to the logical expression simplification we have done in combinational circuit design. We can use any of the methods discussed in Chapter 2. Here, we use the Karnaugh map method to simplify the logical expressions for the six inputs of the three JK flip-flops. Figure 4.18 shows the six Karnaugh maps for the JK inputs. Note that the cells in these Karnaugh maps represent the present state values for A, B, and C.

As a final step, we write the logic circuit diagram based on these logical expressions. In addition to the three JK flip-flops, we just need a 2-input AND gate to implement the 3-bit synchronous counter as shown in Figure 4.19. Contrast this design with the synchronous counter design shown in Figure 4.15. The main difference is that our new design uses the common clock

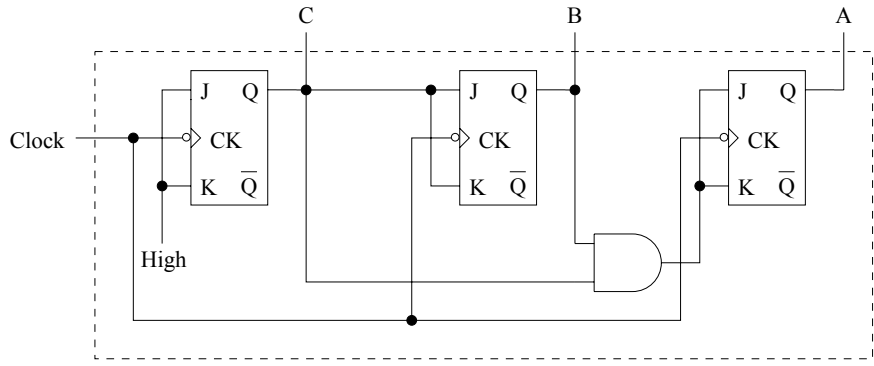

Figure 4.19 Another 3-bit synchronous counter design.

while manipulating the JK inputs. The previous design, on the other hand, manipulates the clock inputs while holding JK inputs high.

A More General Counter Design Example

The previous counter example is simple in the sense that the next state is always an increment of the current state and there is a natural wraparound when the count reaches the maximum value. We know that counters can behave differently. An example we have seen before is the decade counter. In this counter, the next state after 9 is 0. How do we design such counters? We go even a step further and show a counter that jumps states in no specific order. Once we show how such a general counter can be designed, decade counter design becomes a special case of this design process.

For our example, we consider designing a 3-bit synchronous counter with the state transitions shown in Figure 4.20. The counter goes through the following states in a cycle:

$$0 \rightarrow 3 \rightarrow 5 \rightarrow 7 \rightarrow 6 \rightarrow 0.$$

This example counter is different from our binary counter example in certain respects. First, not all states are present. This is similar to the decade counter example, which skips states 10 through 15. The other feature is that next state seems to be an arbitrary state (i.e., not a state that can be obtained by incrementing the current state).

Despite these differences, the design process we have used for the binary counter example applies. The design table to derive the JK inputs is shown in Table 4.5. One significant difference from the table used for the previous counter example and this is the presence of don't care values in the next state columns for states 1, 2, and 4 (shown by a dash in Table 4.5). Because the next state for these three states is a don't care, all JK inputs also assume don't care inputs. This would help us in simplifying the logical expressions for the JK inputs.

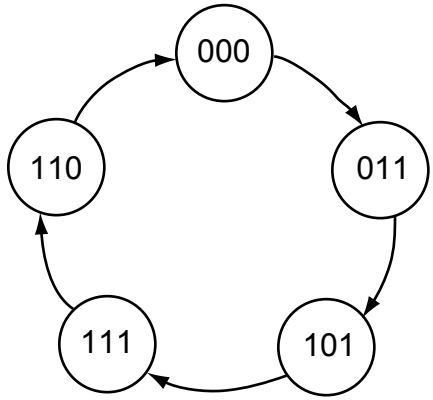

Figure 4.20 State diagram for the general counter example.

Table 4.5 Design table for the general counter example

Present state			Next state			JK flip-flop inputs					
A	B	C	A	B	C	J_A	K_A	J_B	K_B	J_C	K_C
0	0	0	0	1	1	0	d	1	d	1	d
0	0	1	–	–	–	d	d	d	d	d	d
0	1	0	–	–	–	d	d	d	d	d	d
0	1	1	1	0	1	1	d	d	1	d	0
1	0	0	–	–	–	d	d	d	d	d	d
1	0	1	1	1	1	d	0	1	d	d	0
1	1	0	0	0	0	d	1	d	1	0	d
1	1	1	1	1	0	d	0	d	0	d	1

Figure 4.21 shows the Karnaugh maps to derive the simplified logical expressions for the JK inputs. The final logical circuit for this counter is shown in Figure 4.22.

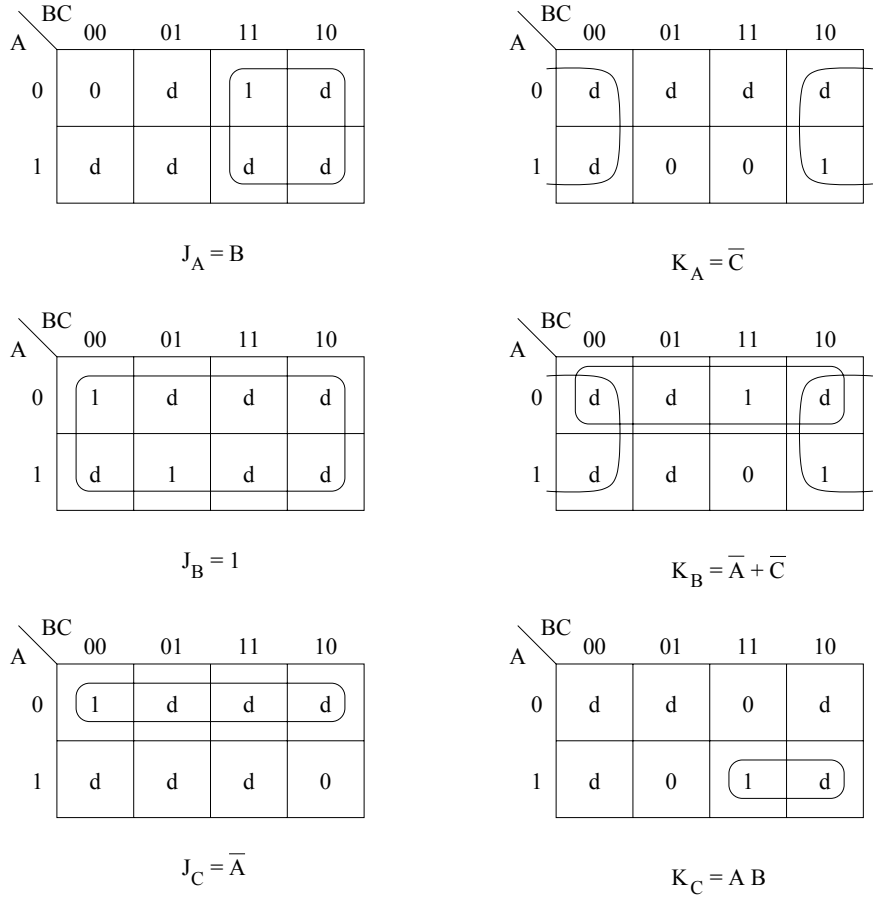

Figure 4.21 Derivation of JK logical expressions for the general counter example.

4.6.2 General Design Process

The counter design example discussed in the last section is a special case of the more general sequential circuit design process. A finite state machine can express the behavior of a sequential circuit. A finite state machine consists of a set of (finite number of) states that the system can take. State transitions are indicated by arrows with labels X/Y, where X represents the inputs that cause the change in the system state and Y represents the output generated while moving from the current state to the next state. Note that X and Y could represent multiple bits.

Example 4.1: *Even-parity checker.* Parity is used to provide rudimentary error detection capability. For example, we can associate a parity bit for each 7-bit ASCII character transmitted. Even parity means that the total number of 1's in the 8-bit group (7 data bits + 1 parity bit)

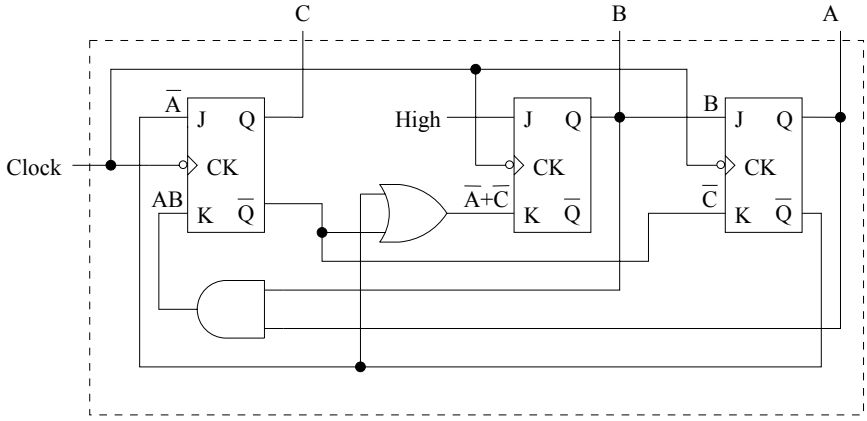

Figure 4.22 Logic circuit for the general counter example.

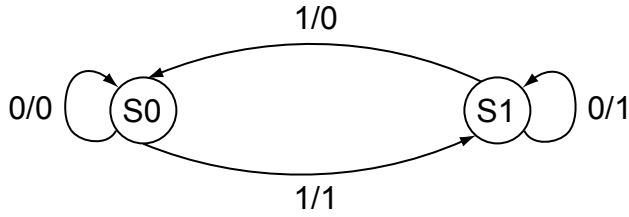

Figure 4.23 State diagram for the even-parity checker.

should be an even number. The receiver of the data also computes the parity and checks if the data have been received properly. For a properly received 8-bit datum, the computed parity bit should be 0. Parity provides single-bit error detection capability.

We have seen in Section 2.10.2 (see Figure 2.27 on page 77) that we can use XOR gates to generate even parity. However, this circuit is not useful for us if the data are coming serially, one bit at a time as on a modem line. If we want to use the XOR implementation, we need to convert the serial input data to parallel form. The other alternative is to design a sequential circuit that receives data in serial form and generates the parity bit.

A simple analysis leads us to the conclusion that the FSM needs to remember only one of two facts summarizing the past input sequence: whether the number of 1's is odd or even. Thus, our FSM needs just two states as shown in Figure 4.23. In the FSM, state S0 represents the fact that the input sequence so far has an even number of 1's. The odd number of 1's is represented by state S1.

Now we have to look at the possible transitions between these two states. When the machine is in state S0, if a 1 is received, the machine should move to S1. Also, it should output a 1 as

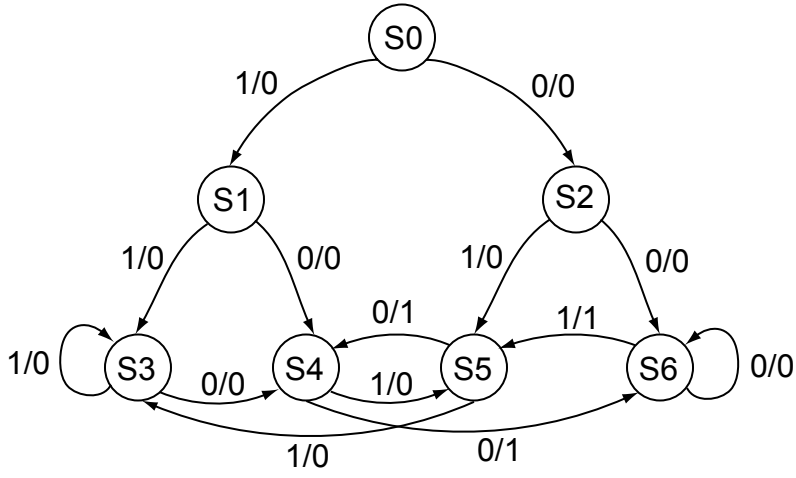

Figure 4.24 State diagram for the pattern recognition problem.

the parity bit for the data it has received so far. This transition is shown in the figure with label 1/1 to represent the fact that the input is 1 and the output is also 1. On the other hand, if a 0 is received in state S0, it remains in that state with output 0.

Similarly, when in state S1, if a 0 is received as input, it remains in state S1 with a 1 as output because the number of 1's in the input so far is odd. A 1 input takes the machine from S1 to S0 with 0 output.

If we want to carry this design forward, we need a single flip-flop to represent the two states in the FSM. We leave the complete design of this machine as an exercise to be done after reading this section (see Exercise 4–13). We show the design process on the next example, which is more complex.

Example 4.2: *Pattern recognition.* As another example, consider designing a system for recognizing a specific bit pattern in the input bit sequence, which enters the system serially. The particular system we wish to design outputs a 1 whenever the input bit sequence has exactly two 0s in the last three input bits.

The system maintains a memory of the last two bits and looks at the next bit coming in to determine the output. For example, the input sequence 010100101001000 produces 101111011111000 as the output sequence. In this example, we assume that the rightmost bit is the first bit to enter the system.

The FSM for this example is shown in Figure 4.24. State S0 represents the initial state (i.e., no input has been received yet). From this state, whether the input is 0 or 1, the output is 0. However, we need to remember the first input bit. Thus, we create two states S1 and S2. From both S1 and S2, whatever the input is, the output is 0 as we have seen only two bits so far. Notice that we visit these three states—S0, S1, and S2—only once during the initial phase.

After receiving the first two input bits, we can be in one of the four states—S3, S4, S5, or S6—depending on the values of these two bits. For example, we will be in S6 if the first two bits of the input sequence are 00. If the input is 01, we will be in state S4. We can summarize the state information captured by these four states as follows:

S3: Last two bits are 11;
S4: Last two bits are 01;
S5: Last two bits are 10;
S6: Last two bits are 00.

Having this summary information helps us in adding the transitions to the FSM among these states. For example, consider state S3. That means the last two bits are 11. Suppose the next bit is 0. The last three bits have two 1's and a 0. Therefore, the output should be 0. And what should be the new state after receiving 0? Since the last two bits are 01, the new state should be S4 as shown in Figure 4.24.

What if, when in state S3, the next bit is 1? In this case, the last three bits are 111 and hence the output should be 0. Next state is the state that represents 11, which is S3 itself. This is shown as a loop with a 1/0 label in the state diagram.

You can apply this method to complete all the transitions among the four states of the FSM that represents the pattern recognition problem.

Steps in the Design Process

Our design process consists of the following steps:

1. *Derive FSM:* From the problem specification, derive a finite state machine representation of the problem. Make sure that the FSM truly and accurately represents the problem specification.
2. *State Assignment:* Assign binary states of the flip-flops to the FSM states. This step is necessary to get an efficient implementation. This problem did not arise in our counter design examples.
3. *Design Table Derivation:* Derive a design table for the state assignment selected in the last step. This step is similar to the design tables used in the design of counters. However, in a general sequential circuit, both input and output may be present.
4. *Logical Expression Derivation:* Derive logical expressions for the JK inputs and the logical expression for the combinational circuit to generate the output. This step is similar to the process described in the last section.
5. *Implementation:* As a final step, implement the logical expressions obtained in the last step.

Since we already derived the FSM for our example, we look at the remaining steps next.

State Assignment

The state diagram for the pattern recognition problem (Figure 4.24) consists of seven states—S0 through S6. That means we need three flip-flops to keep the state information. In general, the number of flip-flops required is $\lceil \log_2 N \rceil$, where N is the number of states in the state diagram. The problem we have now is: How do we assign the state diagram states to the flip-flop states? Of course, we can randomly assign a bit pattern for each state or use a sequence based on the state number. For example, we could assign 000 for S0, 001 for S1, and so on. The problem is that such a state assignment might not lead to an efficient design in that the logic required to drive the JK inputs as well as the logic required to generate the output might not be minimal.

To obtain a good design, we preset three heuristics that help us select a good state assignment. We consider two states adjacent if the number of bit positions in which they differ is exactly one. For example, 000 and 010 are adjacent whereas 010 and 001 are not. If you are familiar with Hamming distance, adjacent states are states with a Hamming distance of one. The three heuristics are summarized below:

1. States that have the same next state for a given input should be assigned adjacent states.
2. States that are the next states of the same state should be assigned adjacent states.
3. States that have the same output for a given input should be assigned adjacent states.

Why do these heuristics make sense? What we are suggesting by these heuristics is that there can be only one variable change between adjacent states. The reason is that these states will end up as neighbors in the Karnaugh map we use later (as we did with the counter examples) to simplify the logical expressions. Having them together simplifies our logical expressions for the JK inputs and the output.

Heuristic 1 says that all input states of a state should be assigned adjacent states so that they form an area in the Karnaugh map. Similarly, all output states of a state should also be assigned adjacent states. In heuristics 1 and 2, we fix the input (i.e., for the same input). These two heuristics are useful in obtaining simplified expressions for the JK inputs. The last heuristic is useful in simplifying the logical expression for the output.

Of course, it is not always possible to satisfy all three heuristics simultaneously. We try our best in assigning adjacent states by giving priority, for example, to the frequency of adjacency requirement.

We illustrate the process of state assignment with the pattern recognition example. We can write the state table, shown in Table 4.6, from the state diagram of Figure 4.24. We can apply our three heuristics to this table. Application of heuristic 1 suggests that states S1, S3, and S5 should be assigned adjacent states as they all have S4 as the next state when the input is 0 (i.e., X = 0). Similarly, when X = 0, S2, S4, and S6 should be assigned adjacent states as they all have S6 as their next state. In our example, we get the same groupings even when X = 1. To indicate that each group is applicable twice, we use superscript 2. When there is a state assignment conflict, we use this weight to resolve the conflict.

Heuristic 1 Groupings: (S1, S3, S5)^2 (S2, S4, S6)^2.

Table 4.6 State table for the pattern recognition example

	Next state		Output	
Present state	X = 0	X = 1	X = 0	X = 1
S0	S2	S1	0	0
S1	S4	S3	0	0
S2	S6	S5	0	0
S3	S4	S3	0	0
S4	S6	S5	1	0
S5	S4	S3	1	0
S6	S6	S5	0	1

Applying Heuristic 2 leads us to the following groupings:

Heuristic 2 Groupings: (S1, S2) $(S3, S4)^3$ $(S5, S6)^3$.

Notice that the (S3, S4) group occurs three times: states S1, S3, and S5 lead to the same set of states. Similarly, states S2, S4, and S6 lead to the (S5, S6) group.

When applying Heuristic 3, if the output is a single bit, it is sufficient to group the states with a 1 output. Thus, we have

Heuristic 3 Groupings: (S4, S5).

We can use a Karnaugh map to get an assignment that satisfies as many adjacency requirements as possible. The Karnaugh map used for the state assignment is shown in Figure 4.25. We start with the assignment of 000 to S0. This initial assignment is arbitrary. From the groupings, we see that there is a strong requirement for assigning adjacent states for S3 and S4 as well as for S5 and S6. Each occurs three times in the groupings obtained by applying Heuristic 2. S1 and S2 should also be assigned adjacent states.

For each assignment, we can use a column in the Karnaugh map to satisfy the adjacency requirement. With this information, the assignment can be arbitrary within each column. Furthermore, placement of these columns in the Karnaugh map can be done arbitrarily. However, if we look at the groupings obtained with Heuristic 1, we see that S1, S3, and S5 should be adjacent. Similarly, S2, S4, and S6 should be adjacent. Well, we cannot satisfy this requirement completely. We can only have two of these three states adjacent. Although this additional

A \ BC	00	01	11	10
0	S0	S3	S5	S1
1		S4	S6	S2

Figure 4.25 Karnaugh map for the state assignment.

Table 4.7 State assignment

State		A	B	C
S0	=	0	0	0
S1	=	0	1	0
S2	=	1	1	0
S3	=	0	0	1
S4	=	1	0	1
S5	=	0	1	1
S6	=	1	1	1

information restricts our state assignment within a column, it still gives us freedom as to the placement of columns relative to S0. Heuristic 3 suggests that S4 and S5 should be assigned adjacent states. We cannot satisfy this requirement as the adjacency requirements from Heuristics 1 and 2 are much stronger.

This leads us to the final assignment shown in Table 4.7. Note that there are other equally good state assignments for this example. In Exercise 4–14, you are asked to experiment with other assignments.

Remaining Design Steps

The remaining steps mentioned in our design process are similar to the design process we have used in the counter design examples. The only difference is that we have to add the inputs and outputs to the design table as shown in Table 4.8. This table has three groups of variables. The first group consists of the current state information A B C and the current input X. Because we have a single bit input, we create two rows with the same A B C values, but with a different

Table 4.8 Design table for the pattern recognition example

Present state			Present state	Next state			Present state	JK flip-flop inputs					
A	B	C	X	A	B	C	Y	J_A	K_A	J_B	K_B	J_C	K_C
0	0	0	0	1	1	0	0	1	d	1	d	0	d
0	0	0	1	0	1	0	0	0	d	1	d	0	d
0	0	1	0	1	0	1	0	1	d	0	d	d	0
0	0	1	1	0	0	1	0	0	d	0	d	d	0
0	1	0	0	1	0	1	0	1	d	d	1	1	d
0	1	0	1	0	0	1	0	0	d	d	1	1	d
0	1	1	0	1	0	1	1	1	d	d	1	d	0
0	1	1	1	0	0	1	0	0	d	d	1	d	0
1	0	1	0	1	1	1	1	d	0	1	d	d	0
1	0	1	1	0	1	1	0	d	1	1	d	d	0
1	1	0	0	1	1	1	0	d	0	d	0	1	d
1	1	0	1	0	1	1	0	d	1	d	0	1	d
1	1	1	0	1	1	1	0	d	0	d	0	d	0
1	1	1	1	0	1	1	1	d	1	d	0	d	0

X value. For example, see the first two rows: the first row is used to indicate the next state transition when A B C = 000 and the input X = 0; the second row identifies the next state if the input is 1 instead. The number of rows we add with the same current state value is 2^n, where n is the number of bits in the input.

The second group consists of the next state values and the current output. This output is the one associated with the transition from the current state to the next. The third group consists of the JK inputs as in the counter examples. These JK input values are filled using the excitation table for the JK flip-flop (see Table 4.3*b* on page 128).

We use the Karnaugh map method to derive simplified expressions for the JK inputs. The Karnaugh maps along with the simplified logic expressions for the JK inputs of the three flip-flops are shown in Figure 4.26. At this point, we have all the information to design the feedback

circuit of the final sequential circuit. Figure 4.28*a* shows this circuit consisting of three JK flip-flops.

To complete the design of the sequential circuit, we need to design the combinational circuit that generates the output Y. We again use a Karnaugh map whose cells are filled with values from the Y column in Table 4.8. The Karnaugh map along with the simplified expression for Y is shown in Figure 4.27. The logical expression for Y

$$Y = \overline{A}BC\overline{X} + ABCX + A\overline{B}\,\overline{X}$$

can be rewritten as

$$Y = BC(\overline{\overline{A}X + A\overline{X}}) + A\overline{B}\,\overline{X}.$$

An implementation of this expression is shown in Figure 4.28*b*.

Figure 4.28 clearly shows the two main components—the combinational logic circuit and the sequential feedback circuit—of the sequential circuit block diagram shown in Figure 4.1 on page 110.

4.7 Summary

In the last two chapters, we have focused on combinational circuits. In combinational circuits, the output depends only on the current inputs. In contrast, output of a sequential circuit depends both on the current inputs as well as the past history. In other words, sequential circuits are state-dependent whereas the combinational circuits are stateless. The state-dependent behavior of a sequential circuit can be described by means of a finite state machine.

Design of a sequential circuit is relatively more complex than designing a combinational circuit. In sequential circuits, we need a notion of time. We have introduced a clock signal that provides this timing information. Clocks also facilitate synchronization of actions in a large, complex digital system that has both combinational and sequential circuits.

We have discussed two basic types of circuits: latches and flip-flops. The key difference between these two devices is that latches are level sensitive whereas flip-flops are edge-triggered. These devices can be used to store a single bit of data. Thus, they provide the basic capability to design memories. We discuss memory design in Chapter 16.

We have presented some example sequential circuits—shift registers and counters—that are commonly used in digital circuit design. There are several other sequential circuit building blocks that are commercially available.

We have given a detailed description of the sequential circuit design process. We have presented the design process in two steps. To develop your intuition, we have discussed a simple counter design using JK flip-flops. We have then presented a more complex, general sequential circuit design process by using two examples.

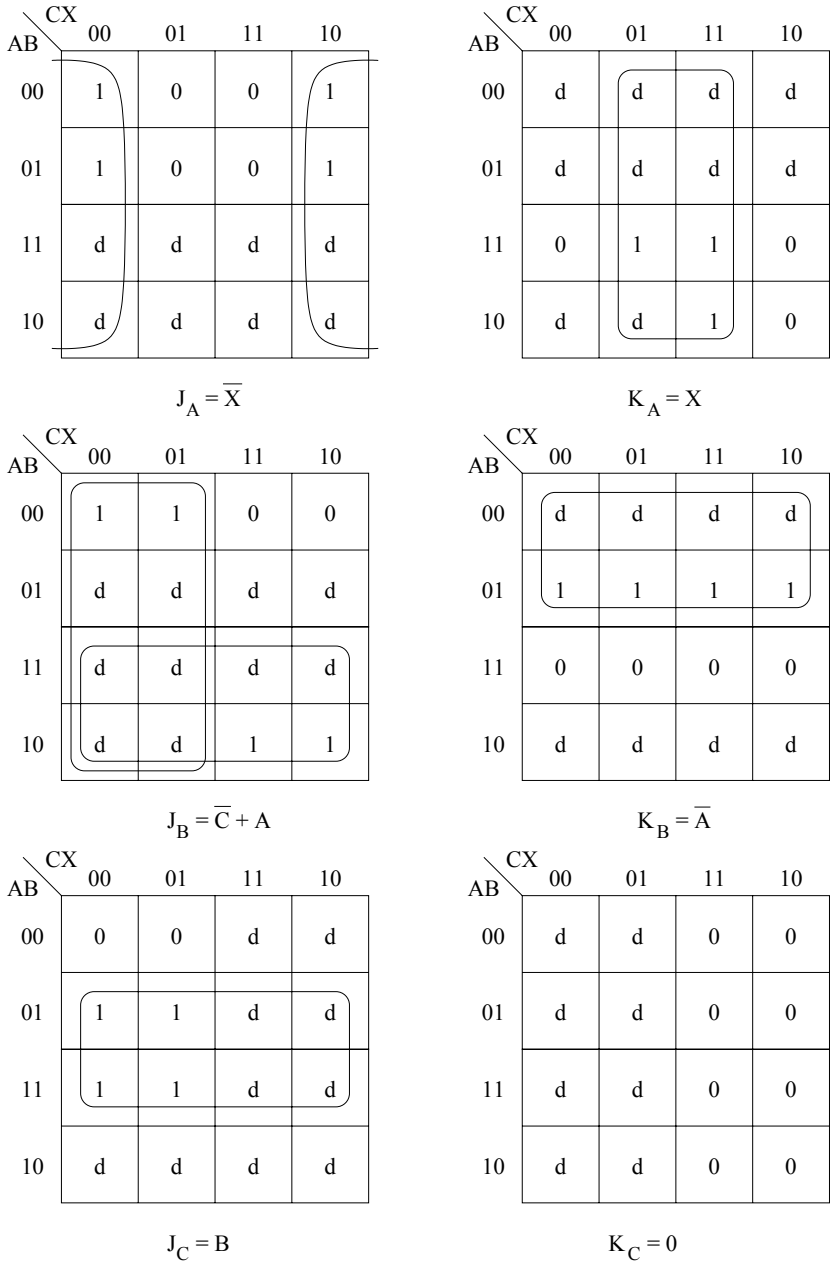

Figure 4.26 Karnaugh maps for the JK inputs.

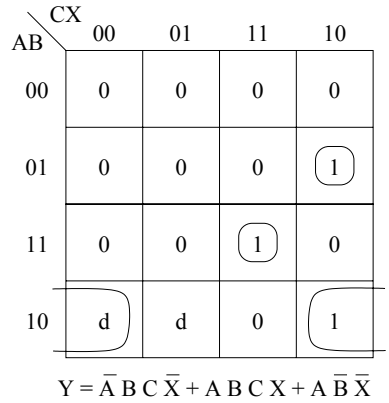

Figure 4.27 Karnaugh map for the output.

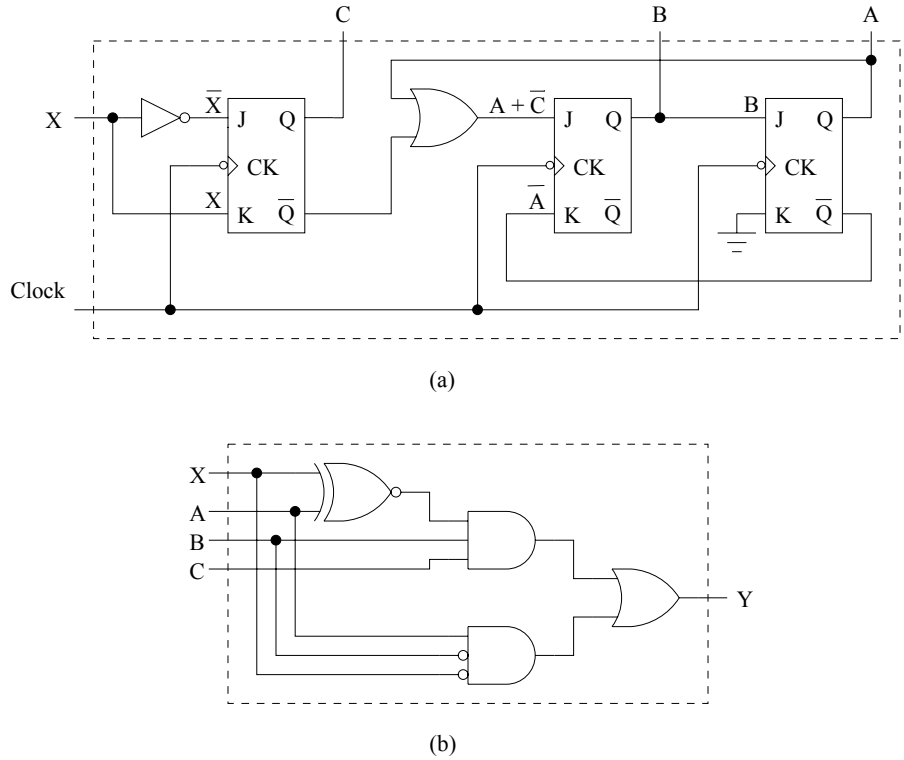

Figure 4.28 Sequential circuit for the pattern recognition example.

Key Terms and Concepts

Here is a list of the key terms and concepts presented in this chapter. This list can be used to test your understanding of the material presented in the chapter. The Index at the back of the book gives the reference page numbers for these terms and concepts:

- Binary counter design
- Clock cycle
- Clock frequency
- Clock period
- Clock signal
- Clocked SR latch
- Counters
- D flip-flops
- D latch
- Falling or trailing edge
- Flip-flops
- General counter design
- JK flip-flops
- Latches
- Rising or leading edge
- Sequential circuit design steps
- Shift registers
- SR latch

4.8 Exercises

4–1 What is main difference between a combinational circuit and a sequential circuit?

4–2 What is the purpose of the feedback circuit in a sequential circuit?

4–3 If the propagation delay is zero, what is the output of the circuit shown in Figure 4.4a?

4–4 Explain the reason for not allowing the input combination S = 1 and R = 1 in Figure 4.5.

4–5 We have shown an implementation of an SR latch in Figure 4.5a using two NOR gates. Suppose we replace the NOR gates by NAND gates. Give the truth table for this new circuit. Can you argue that this new circuit is essentially equivalent to the one in Figure 4.5a?

4–6 How is the D latch avoiding the input combination S = 1 and R = 1?

4–7 What is the difference between a latch and a flip-flop?

4–8 Using the 7476 JK flip-flop chips, construct a 4-bit binary counter. This circuit should have a reset input to initialize the counter to 0.

4–9 Extend the 3-bit synchronous counter shown in Figure 4.15 to implement a 5-bit synchronous counter.

4–10 Extend the 3-bit synchronous counter shown in Figure 4.19 to implement a 4-bit synchronous counter.

4–11 Suppose that we are interested in a 4-bit binary counter that counts through only the even values. That is, the counter starts at 0 and goes through $0 \rightarrow 2 \rightarrow 4 \rightarrow 6 \rightarrow 8 \rightarrow 10 \rightarrow 12 \rightarrow 14 \rightarrow 0$ and so on. Design such a counter using JK flip-flops.

4–12 In this exercise, implement a 3-bit *gray code* counter. A gray code is a sequence of numbers in which successive numbers differ in only one bit position. An example 3-bit gray code sequence is: 000, 001, 011, 010, 110, 111, 101, and 100. The code is cyclic,

which means that the sequence repeats after 100. Design a counter to implement this gray code sequence.

4–13 We have used the even-parity example to illustrate the general sequential circuit design process. But we did not complete the design (see Example 4.1 on page 132). In this exercise, complete the design of this circuit.

4–14 We have discussed several heuristics to assign states on page 136. Try a different state assignment (possibly the worst) and carry the design through. Compare the final design of your assignment with the one shown in Figure 4.28.

4–15 We have designed a sequential circuit to recognize exactly two zeros in the last three input bits. Can you easily modify the sequential circuit to recognize exactly two ones in the last three inputs?

4–16 Design a sequential circuit that recognizes the bit pattern 010 in an input bit stream. This circuit outputs a 1 whenever the pattern 010 appears. Here is an example input bit stream and the corresponding output of the sequential circuit:

Input: `11101010010101010101`;
Output: `00010100101010100100`0.

As in our pattern recognition example on page 134, assume that the rightmost bit is the first bit to enter the system.

4–17 This is a variation on the last exercise. We are interested in identifying the bit pattern 010 but not on a continuous basis. Once a 010 pattern is recognized, it skips these three bits and looks for another 010 pattern as shown in the following example:

Input: `11101010010101010101`;
Output: `00000100000100010010`00.

Design a sequential circuit for this problem.

4–18 Consider a vending machine that accepts nickels (5 cents) and dimes (10 cents). All products cost 25 cents. The machine requires exact change (i.e., it will not give out change). Design a sequential circuit that accepts 25 cents and activates a selection circuit. Activating the selection circuit enables the user to select an item from the selection panel. Since the machine will not give change, if someone gives three dimes, it will allow the user to select an item (and keeps the extra 5 cents). *Hint:* You can represent the input using two bits: one for the nickel and the other for dime. Then the input combination 01 represents insertion of a nickel; 10 represents insertion of a dime. If the input is 00, it means that no coins have been inserted into the machine. Obviously, input 11 cannot occur, as the machine does not allow insertion of two coins at the same time. You can represent the output by a bit (selection circuit activated or not).

4–19 Modify the last exercise to make the machine give the change. Note that you need to take care of the case when someone inserts three dimes. Do not worry if someone inserts six or more nickels. You simply activate the selection circuit after inserting five nickels.